UNITED STATES UTILITY PATENT APPLICATION

FOR

CAUSALITY-BASED MEMORY ORDERING IN A MULTIPROCESSING ENVIRONMENT

Inventor:

Deborah T. Marr

42390.P8011C

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN

12400 Wilshire Boulevard, Seventh Floor
Los Angeles, California 90025-1030
(408) 720-8598

EXPRESS MAIL CERTIFICATE OF MAILING

"Express Mail" mailing label number EV305339434US
Date of Deposit December 2, 2003

I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service on the date indicated above and is addressed to Mail Stop Patent Application, Commissioner for Patents, PO Box 1450, Alexandria, VA 22313-1450.

Anne Collette
(Typed or printed name of person mailing paper or fee)

Anne Collette 12/2/2003
(Signature of person mailing paper or fee) Date

CAUSALITY-BASED MEMORY ORDERING IN A MULTIPROCESSING ENVIRONMENT

CROSS REFERENCES TO RELATED APPLICATIONS

This is a continuation of application Ser. No. 09/474,527, filed December 29, 1999, currently pending.

BACKGROUND

1. Field

The present disclosure pertains to the field of processing systems. More particularly, the present disclosure pertains to a memory ordering technique for a multiprocessing system.

2. Description of Related Art

Improving the performance of computer or other processing systems generally improves overall throughput and/or provides a better user experience. One technique of improving the overall quantity of instructions processed in a system is to increase the number of processors in the system. Implementing multiprocessing (MP) systems, however, typically requires more than merely interconnecting processors in parallel. For example, tasks or programs may need to be divided so they can execute across parallel processing resources.

Another major challenge in an MP system is maintaining memory consistency (also known as coherency). Memory consistency is the general requirement that memory remain

sufficiently updated to supply a current copy of memory contents to a requesting processor or other device. Maintaining memory consistency is complicated by the use of internal caches and other data structures that store data for more efficient access than is typically available from other (e.g., external) memory circuits.

A system may maintain memory consistency using hardware or using a combination of hardware and software techniques. The hardware provides a particular memory ordering guarantee, a guarantee that the hardware will maintain the sequential nature of program memory accesses (to at least some selected degree) at some selected point in the system hierarchy. Software may be used in some systems to supplement hardware-provided memory ordering by forcing additional ordering restrictions at desired times. The memory ordering scheme implemented is a design choice involving a tradeoff between hardware complexity, software complexity, and the desired ability to cache and buffer data.

One prior art technique that represents a compromise between weakly ordered memory consistency models and very restrictive consistency models is "processor consistency". The processor consistency model is a known prior art model which allows limited reordering. One implementation is used in some prior current processors (see, e.g., US Patent 5,420,991). Memory ordering constraints for one embodiment of a prior art processor consistency memory model system are shown in Figure 1a.

According to block 100 of Figure 1a, the prior art system ensures that stores from each individual processor in the system are observed in order by all other processors. In other words, individual stores from a particular processor are not re-ordered with respect to each other. As indicated in block 102, the system ensures that loads from each

processor appear to execute in order. In some systems, optimizations may be done; however, load data appears to be returned to the computation-performing unit in order to avoid altering the ordering relationships between the system loads and stores. On the other hand, if the load data being returned has not been altered by non-globally-observed stores, the order of that the load data is returned may be varied, and the data still appears to be returned in order.

Additionally, as indicated in block 104, the system ensures that loads and stores to the same address are globally ordered. Thus, all agents in the system observe loads and stores to the same address in the same order. The consequences of the constraints of blocks 100-104 are discussed in greater detail (see Figures 4a-b) as some embodiments of the present invention include these constraints as well.

Finally, as indicated in block 105, stores to different addresses by different processors are globally ordered except that each processor can observe its own stores prior to observing stores from other processors. This prior art constraint is further contrasted with the present invention below (see Figures 4c-4e for implications of this prior art constraint). Some systems (e.g., systems based on the Profusion Chipset from Intel Corporation of Santa Clara) may require substantial hardware to ensure reasonably efficient ordered global observation of different stores to different memory locations by different processors.

Moreover, memory ordering overhead continues to grow dramatically as systems which implement traditional memory ordering models are scaled up to meet additional processing challenges. Consequently, there is a continuing need for memory ordering

techniques that allow improved efficiency while maintaining a predetermined memory ordering protocol such as processor consistency.

Brief Description of the Figures

The present invention is illustrated by way of example and not limitation in the Figures of the accompanying drawings.

Figure 1a illustrates a prior art memory ordering technique.

Figure 1b illustrates one embodiment of an alternative memory ordering technique.

Figure 2a illustrates one embodiment of a system capable of operating according to disclosed memory ordering techniques.

Figure 2b illustrates one embodiment of a technique for store forwarding which may be utilized by the system in Figure 2a.

Figure 3a is a flow diagram illustrating operations for processor A, processor B, and the arbitration logic according to one embodiment when a causal relationship exists between stores.

Figure 3b is a flow diagram illustrating operations for processor A, processor B, and the arbitration logic according to one embodiment when no causal relationship exists between stores.

Figures 4a, 4b, 4c, 4d, 4e, 4f, 4g and 4h illustrate exemplary memory access sequence occurring in one embodiment of a system using disclosed memory ordering techniques.

Figure 5 illustrates one embodiment utilizing a switch-based multiprocessing architecture.

Figure 6 illustrates one embodiment utilizing a hierarchical cluster-based architecture.

5

Detailed Description

The following description provides causality-based memory ordering in a multiprocessing environment. In the following description, numerous specific details such as system arrangements and hierarchies, types of bus agents, and logic partitioning/integration choices are set forth in order to provide a more thorough understanding of the present invention. It will be appreciated, however, by one skilled in the art that the invention may be practiced without such specific details. In other instances, control structures and gate level circuits have not been shown in detail in order not to obscure the invention. Those of ordinary skill in the art, with the included descriptions, will be able to implement the necessary logic circuits without undue experimentation.

The disclosed memory ordering techniques may advantageously improve overall processing throughput in some systems. Improved throughput may be achieved by relaxing traditional memory ordering rules in a manner that allows the system to more efficiently order load and/or store accesses. Efficiency may be improved by either actively reordering memory accesses or by allowing memory accesses to occur in a new order permitted by the relaxed ordering rules. Such memory ordering rules may be implemented so that processor consistency is maintained, thereby allowing backwards compatibility with software that assumes traditional processor consistency is provided. "Processors" may be bus agents or devices of any type which process data and/or instructions and therefore need to access memory.

Figure 1b illustrates one embodiment of a disclosed memory ordering technique. Various details and embodiments are included in the remaining Figures and accompanying description further explain the memory ordering techniques referred to in Figure 1b. Blocks 100-104 of Figure 1b also appear in and are discussed with respect to Figure 1a.

The presently disclosed techniques deviate from the restrictive ordering constraints detailed in block 105 of Figure 1a. As indicated in block 106 of Figure 1b, causal relationships are maintained. Maintaining causal relationships entails maintaining sufficiently sequential access to obtain correct results in cases where one access affects or may affect another. In other words, if a value stored by a first store from a first processor is loaded by a second processor, the ordering of these two memory accesses with respect to subsequent stores from the second processor is important. Subsequent stores by the loading processor should be ordered after the load and the first store since the first store may affect the value stored by the second store.

A true causal connection between two stores executed by different processors occurs when the second store from the second processor directly depends on the value generated by the first store from the first processor. Due to speculative execution and aggressive prefetching techniques, among other things, it may be difficult to precisely determine when the two stores are truly causally linked. Accordingly, for simplicity, causality may be assumed when the second processor merely loads the value stored by the first processor in the first store.

Thus, observation of a particular memory location occurs when an agent loads all or a portion of the contents of that memory location. "Global observation" of a newly

stored value is achieved when data has propagated sufficiently through the system that potential observers will observe the new value if they load the affected memory location. In other words, all agents would see the new value if they performed a load operation after global observation of the store. Local observation occurs when a subset of the potential observers would see the new value if they performed a load.

Block 110 further indicates a relaxation of traditional memory ordering rules according to the presently disclosed techniques. Block 110 states that the system reorders a set of stores generated in a first order by more than one processor to be observed in a different order. In other words, under certain circumstances, stores from a first processor may be re-ordered with respect to stores from another processor. Stating this in the negative, the system does not force global ordering of stores to different addresses by different processors. In some embodiments, different processors may observe such sets of stores in different orders. Notably, the "first order" that the set of stores are in may include simultaneous stores as multiple processors in some multiprocessing systems may generate stores at the same time.

The re-ordering allowed in block 110, however, is subject to several conditions according to one embodiment. In this embodiment, the re-ordering performed in block 110 is subject to the constraints of block 100, 102, 104, and 106. Namely, inter-processor store ordering may be altered as long as stores from each individual processor remain in order, loads from each processor appear to execute in order, loads and stores to the same address are globally ordered, and causality is maintained. Other embodiments may employ only a subset of these restrictions in implementing a multiprocessing system with causality based memory ordering.

Figure 2a illustrates one embodiment of a system that implements disclosed causality-based memory ordering techniques. The system of Figure 2a has a first plurality of processors including processor A 205 and processor B 210 coupled to cluster 1 arbitration logic 230, causality monitoring logic 232, access optimization logic 234 and buffering logic 236. Similarly, the system includes processor C 215 and processor D 220 coupled to cluster 2 arbitration logic 240, causality monitoring logic 242, access optimization logic 244, and buffering logic 246.

The cluster arbitration logic 230 and 240 is coupled to central arbitration logic 250, causality monitoring logic 260, access optimization logic 270, and buffering logic 280. Each of these sets of arbitration logic, causality monitoring logic, access optimization logic, and buffer logic may function similarly. In one embodiment, the various logic elements cooperate to implement the memory ordering protocol indicated in Figure 1b. Specifically, the access optimization logic blocks may improve the efficiency of the completion order for loads and/or stores buffered by the buffering logic unless the causality monitoring logic indicates that such reordering is problematic. The central arbitration logic 250 may coordinate these activities, perform its own optimizations, and ultimately dispatch memory accesses to a memory (not shown) or other arbitration logic (not shown).

The details of these logic various blocks will be discussed with respect to cluster 1; however, a similar implementation may be used for the other similarly labeled blocks. With respect to cluster 1, the buffering logic 236 buffers loads and stores from each agent. The arbitration logic 230 arbitrates and schedules the accesses from all agents in cluster 1 in an efficient manner with the assistance of the access optimization logic 234.

The arbitration logic 230 also arbitrates for resources at the next level in the system hierarchy (e.g., from the central arbitration logic 250).

The access optimization logic 234 performs operations such as access reordering, write combining (WC) , and data forwarding (FWD) to improve the overall efficiency of the accesses generated by arbitration logic. Such optimizations, however, may be limited by the constraints discussed with respect to Figure 1b. Accordingly, the access optimization logic cooperates with the causality monitoring logic 234 to determine the degree of optimization that is permissible. For example, in accordance with block 100 in Figure 1b, stores from each agent are scheduled in order with respect to other stores from that agent.

One optimization that may be performed by the access optimization logic 234 is store forwarding. One embodiment performs store forwarding as detailed in the flow chart of Figure 2b. In general, store forwarding (also known as data forwarding) involves checking received loads against pending (buffered) stores to detect forwarding conditions that allow data buffered with the stores to be forwarded to loads. One forwarding condition arises when a load requests data from a memory location for which there is a previous store from the same agent to the same memory location buffered in the buffering logic 236. As indicated in blocks 281 and 282 of Figure 2b, in this case, the load may be satisfied by forwarding the buffered data to the requesting agent.

As detected block 284, if a load requests data from a memory location for which there is a store from a different agent to the same memory location and (as detected in block 288) there is no causal relationship between the stores of the two agents, then the store data can be forwarded to the load as indicated in block 290. In this case, a causal

relationship is established between the two agents as indicated in block 292. Consequently, the stores prior to the store that provided the forwarded data are to precede the subsequent stores from the agent that received the new data in this embodiment.

If a causal relationship is detected in block 288, then the order of the load and store is checked as indicated in block 294. This occurs when a load requests data from a memory location for which there is a store from a different agent (to the same memory location), and there is a causal relationship between the stores of the two agents. As indicated in block 298, the store data can be forwarded to the load only if the store is ordered previous to the next older store after the load. Otherwise data is not forwarded as indicated in block 296. Thus, store forwarding may be performed more aggressively than in traditional systems, and more efficient overall memory access may advantageously result.

Additionally, general load and store reordering may be done more aggressively than in traditional processor consistency multiprocessing systems because the constraints imposed by the causality-based memory ordering system of Figure 1b are less onerous than that typically used. Thus, more critical memory accesses may be ordered earlier in some cases, and/or overall ordering may be improved. For example, write cycles may be combined or re-arranged to optimize for a particular burst order, interleaving scheme, or other order-related constraint particular to the system involved.

The cluster arbitration logic 230 and 240, the central arbitration logic 250, as well as the causality monitoring logic, the access optimization logic, and the buffering logic for each, may reside in a single integrated circuit, component, module, or the like, as indicated by their inclusion in an arbitration block 200. Alternatively, these logic blocks

may be distributed into various components. One advantageous embodiment is an integrated (e.g., single integrated circuit) multiprocessing system that includes multiple processors (e.g., A, B, etc.) and cluster arbitration logic for that set of processors. Additionally, while the separation of these logical units in two different blocks may aid in an understanding of the presently disclosed techniques, such a clean separation of the various logic functions implemented by these and other blocks discussed herein is not required and indeed may not exist in many implementations.

Figure 3a illustrates operations performed by one embodiment of the system of Figure 2a in one case where a causal relationship exists between stores. As indicated in block 310, Processor A stores values to memory locations a and b. According to block 100 (Figure 1), all stores from processor A are kept in order with respect to each other. Loads, on the other hand, may be re-ordered with respect to some stores in a known or otherwise available manner.

Additionally, according to presently disclosed techniques, store optimizations may be performed at the level of the arbitration block 200. As indicated in block 322, processor B stores a value in memory location x. Since the operation in block 322 by processor B is not causally linked to the stores to locations a and b generated by processor A, the arbitration block 200 can reorder stores generated by processor A and processor B into a convenient order as indicated in block 315. For example, some or all of the writes may be combined or pipelined and/or the writes may be ordered in a sequential order.

As indicated in block 325, processor B may prevent or interrupt such reordering. In block 325, processor B observes (e.g., by loading) memory location b. This observation is recognized by the arbitration block 200, which may continuously check for

accesses to addresses of buffered stores. As indicated in block 330, the observation of memory location b by processor B causes the arbitration block 200 to order stores generated after the observation of processor A's store to b so that such stores are actually completed after the store to location b. Thus, the arbitration block 200 can not reorder the stores to memory location y generated by processor B in block 327 before the store by processor A to location b. This tightened store ordering is enforced, as indicated in block 335, until global observation of b is achieved. Thereafter, store reordering may be resumed as indicated in block 340.

Thus, in the example of Figure 3a, the disclosed memory ordering technique may allow improved efficiency for the case where there is no causal connection between stores generated by different processors. In other words, since the store to memory location a by processor A has no causal relation to store to memory location x by processor B, the stores may be reordered in a manner that is more efficient. On the other hand, since the store to location y by processor B is causally related to the store by processor A to location b, the stores may not be reordered completely. The legal or permissible orders without violating causality are indicated by reference number 345.

Permissible Orderings for Embodiment of Figure 3a

a, b, x, y

a, x, b, y

x, a, b, y

Additional efficiencies may be expected by performing reordering across more

than two processors as a larger number of processors are likely to have more causally unrelated stores.

Figure 3b illustrates operations performed by one embodiment of the system of Figure 2a in one case where no causal relationship exists between stores. As indicated in block 350, processor A performs stores to locations a and b. Similarly, processor B performs stores to locations x and y as indicated in block 355. Since neither processor loaded another value modified by the other processor, there is no causal relationship between the various stores. Thus, as indicated in block 360, no causal relationship is detected by the arbitration block 200. Accordingly, any convenient order may be used to perform the stores, so long as other system constraints are observed.

Permissible Orderings for Embodiment of Figure 3b

a, b, x, y

x, y, a, b

x, a, y, b

a, x, b, y

x, a, b, y

a, x, y, b

Figures 4a-4h illustrate various consequences of using the disclosed memory ordering techniques in a multiprocessing system. In one embodiment, the constraints shown in Figure 1b are enforced.

1. Stores from each processor are observed in order (Block 100).

Again, a store is "observed" when a processor or agent (either the same agent or a different agent) loads a memory location and retrieves the new value of the memory location as modified by the store. When stores from each processor are observed in order, no agent can observe the new value of a sequentially later store from that processor and the old value of a sequentially previous store from that processor.

Figure 4a depicts agent A performing two stores. The initial condition is that all memory locations store 0 (in all of Figures 4a-4h). First, the value 1 is stored in memory location a by agent A. Thereafter, agent A also stores the value 1 to memory location b. Agents B, C, and D observe the two stores in all the legal scenarios (i.e., permissible orderings according to the implemented memory ordering scheme). Agent B observes the new value of a, but the old value of b, which is acceptable because the store to a occurred before the store to b. Agent C observes the old values of both a and b. Agent D observes the new values of both a and b. No agent can observe the new value of b and the old value of a because that would violate the constraint of block 100.

2. Loads from each processor appear to execute in order (Block 102).

In this embodiment, loads are limited to appearing to execute in order. By appearing to execute in order, it is meant that the load data is returned to the requesting computational unit in order (although data transfer signals may be reordered due to bus protocols, transfer techniques, etc.) if non-globally observed stores might affect the load

data. If the load data being returned has not been altered by non-globally-observed stores, then the order of its return may be reordered, and the data still appears to be returned in order since the reordering does not affect the computational unit. If load reordering was allowed for memory locations altered by non-globally-observed stores, then the relative ordering of loads and stores could be affected, and the stores may be observed to execute out of order. In Figure 4a, agents B, C, and D all execute their loads in order and the proper values are returned to the agents. As an example, if one of the agents were allowed to retire loads out of order, b might be observed as changing before a.

3. Loads and stores to the same address are globally ordered (Block 104).

Loads and stores to the same address are globally ordered such that if two agents both store values to the same address, all observing agents observe one of the stores happening before the other. In Figure 4B, agents A and B both store to location a, with agent A storing 1 and agent B storing 2. Agents C, D, and E observe the two stores in all of the possible orders. Agent D observes agent A's store which changed the value of a to 1, and then observes agent B's store which changed the value of a from 1 to 2. Therefore, no other agent could observe the opposite order in this embodiment. Accordingly, agent C can observe a being 0 and then a being 1, and agent E can observe a being 0 and then a being 2. No agent can first observe a being 2 and then a being 1 according to this embodiment.

On the other hand, in Figure 4C, agent D observes Agent B's store which changed the value of location a from 0 to 2, and then observes Agent A's store to the same

location, changing the value from location a from 2 to 1. If another agent saw the opposite order, the constraint in block 104 would be violated. Accordingly, agent C first observes a being 0 and then observes a being 1 (no observation when a is 2). Agent E observes a being 0 and then a being 2.

In prior art "processor consistency" systems (e.g., Figure 1a) , stores to different addresses by different processors are generally globally ordered (block 105). In such prior art systems, stores to different addresses are globally ordered except for the case that each agent can observe its own store prior to observing stores from other processors. In Figure 4D, agents A and B store the value 1 in memory locations a and b respectively. Agent D observes a being 1 before b being 1. Therefore, no other agent observes b being 1 before a being 1 in this embodiment. Agent C observes both a and b being 0, and Agent E observes both a and b being 1.

In Figure 4E, agents A and B again each store the value 1 in memory locations a and b respectively. Agent D observes b being a 1 before a being a 1. Consequently, no other agent observes a being a 1 before b being a 1 in this embodiment. Agent C observes both b and a being 0, and agent E observes b and a being 1.

The exception is shown in Figure 4F, where agents are observing their own stores. In this case, the constraint is not imposed that stores to different addresses need to be globally ordered. Thus, as illustrated in Figure 4F, agent A observes its own store to location a before agent B's store to location b. Similarly, agent B observes its own store to location b before agent A's store to location a. Thus, except for the self-observation exception, this prior art policy is quite restrictive with respect to different stores from different processors.

4. Causal relationships are guaranteed (Block 106).

In contrast to prior art systems which forced global ordering of stores to different addresses by different processors, some embodiments employing presently disclosed techniques only enforce a minimal amount of store ordering as required to maintain causality. Figure 4G illustrates a case where a causal connection is created by observation of a stored value. Agent A stores the value 1 to memory location a, and agent B observes the store to a and then performs a store of the value 1 to memory location b. Agent B introduces a causal relationship between the stores to memory locations a and b, so all agents are constrained to observe the store to a before the store to b. Agent C does just this, observing the store to memory location a first. Agent D is shown as having observed both stores, but did not observe the store to b before the store to a.

If processor B had not observed the store to memory location a, the system could have re-ordered the global observation of these stores to memory locations a and b. In other words, if processor B had not inquired as to the value of location a, processor C (and the rest of the system) may have been indifferent as to whether a or b became 1 first. Since the store to a was observed by processor B, creating a causal connection, the arbitration logic preserved the a-first, b-second ordering.

5. Stores to different addresses by different processors are not globally ordered (Block 110).

Figure 4H illustrates a situation where Agents A and C store to two different addresses. Agent A stores the value 1 in memory location a, and agent C stores the value 1 in memory location b. Then Agents B and D observe the two stores in two different orders. This ordering is allowed provided that there is no causal relationship between agents A and C (i.e., there is no observation of a store from the other agent prior to global observation of that store). Accordingly, the other agents (the non-store-issuing agents, agents B and D in this example) can observe stores in a different order. This ordering is not allowed in traditional processor ordering schemes.

Figure 5 illustrates one embodiment of a system utilizing a switch-based multiprocessing architecture. In this system, a central switch 500 couples a plurality of processors 510, 512, 514, 516, 518, 520, 522 to a plurality of memory and I/O devices including I/O interfaces 530 and 534 and memory systems 532 and 536. The I/O interfaces 530 and 534 may each provide an interface to one or more I/O devices. The basics of such switch-based multiprocessing arrangements are known in the art.

As distinguished from prior switch-based multiprocessing systems, the embodiment shown in Figure 5 implements a processor consistent memory architecture with an embodiment of the causality-based memory ordering technique illustrated in Figure 1b. Accordingly, the central switch 500 ensures that stores from each individual processor are observed in order by all other processors (block 100). The central switch 500 also allows a set of stores generated in a first order by more than one processor to be observed in a different order provided that causality is not violated. To this end, a plurality of buffers 502 are included in the central switch 500 to buffer stores from the various processors before they are committed to a memory (not shown). Access

optimization logic 506 can reorder the stores from the plurality of buffers 502, perform store forwarding, and/or make other optimizations as long as no other system limitations are implicated and causality monitoring logic 504 does not detect a causal relation and accordingly limit the optimization that may be done by the access optimization logic 506.

Figure 6 illustrates a hierarchical, cluster-based multiprocessing system which also implements disclosed techniques for memory ordering. The system of Figure 6 includes a plurality of processors, processors 650, 652, 654, 656, 658, and 660. Processors 650 and 652, as well as any number of additional processors, form a cluster controlled by a controller 672. Similarly, processors 654 and 656 form a cluster controlled by a controller 674, and processors 658 and 660 form a cluster controlled by a controller 676. Controllers 672 and 674 are coupled to an intermediate level controller 680 which in turn is coupled to a top-level controller 690 that interfaces with a memory (not shown). The controller 676 is also coupled to the top-level controller 690. Many other hierarchical arrangements are possible, including using different numbers of processors per cluster, different numbers of clusters, and a different (or no) division of controllers.

In the system of Figure 6, causality monitoring and buffering may be performed at various levels of the hierarchy. For example, buffering (BUF) and causality monitoring (CM) logic may be included in each of the controllers 672, 674, 676, 680 and 690. Stores may then be passed to higher levels in the hierarchy tagged with any known ordering restrictions imposed by the causality monitoring logic. As a result, different processors in the system may observe stores in different orders. For example, the top-level controller 690 may be able to perform more optimizations than low level controllers such as

controller 672 because the top-level controller 690 has more or at least different stores to rearrange, combine, or otherwise manipulate.

As discussed with respect to other embodiments, system throughput may be improved by more efficiently ordering memory accesses. Such optimizations may include or relate to write combining, paging, interleaving, load bypassing, or other known or available memory access optimization techniques. Embodiments disclosed may allow a processor consistency memory ordering model to be maintained, advantageously providing backwards compatibility with existing code that assumes compatibility with a processor consistency memory ordering model.

Thus, causality-based memory ordering in a multiprocessing environment is disclosed. While certain exemplary embodiments have been described and shown in the accompanying drawings, it is to be understood that such embodiments are merely illustrative of and not restrictive on the broad invention, and that this invention not be limited to the specific constructions and arrangements shown and described, since various other modifications may occur to those ordinarily skilled in the art upon studying this disclosure.